Scheduler Module

Version List			
Versión índex	Date	Author	Description
1.0	15/11/2015	Francisco Javier Quirarte Pelayo	First revisión of scheduler
		David Antonio Díaz Ramírez	

Functional Specification Scheduler Mechanism

Scheduling refers to making a sequence of time execution decisions at specific intervals, this decision that is made is based on a predictable algorithm. An application that does not need its current allocation leaves the resource available for another application's use.

The underlying algorithm defines how the term “controlled” is interpreted, in some instances, the scheduling algorithm might guarantee that all applications have some access to the resource. The Binary Progression Scheduler (BPS) manages the access to the CPU resources in a controlled way.

Tasks Partitioning

Task Partitioning is used to bind a task to a subset of the system's available resources, this binding guarantees that a known amount of resources is always available to the task. Those resources are taken by time-slicing the available processing time, systems that use time-slicing take advantage of the CPU/Core utilization and keeping the CPU/Core occupied which enhance the use of the MCU resources. A processor always have a task to execute even though all the other tasks are idle, when no tasks are executed the processor is running a Background Task

Mask Concept

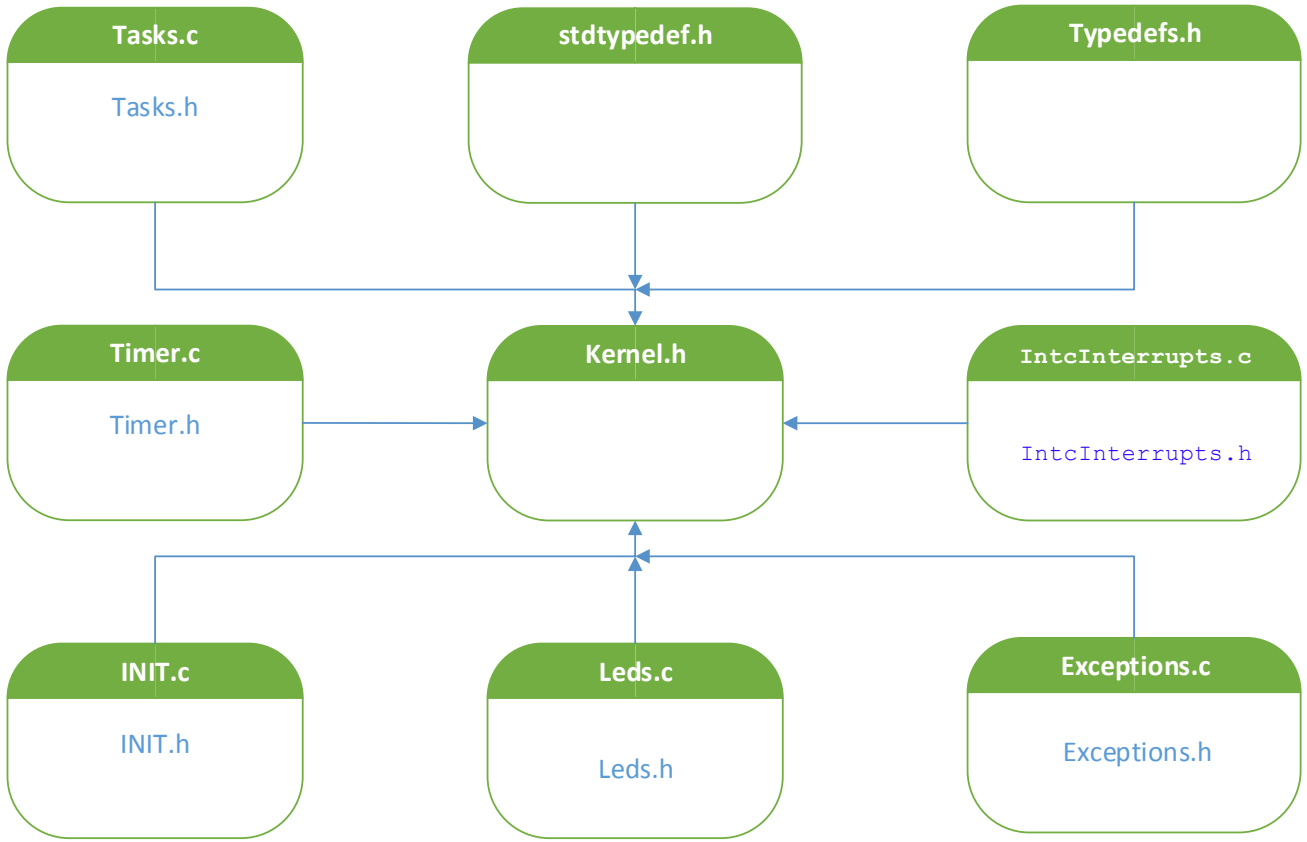
The Scheduler is based on a binary counter incremented at a given time, this time is controlled by a timer interrupt, typically called OS Tick.
A mask is a number defined by: mask = (2^n)-1
Where n represents the counter data size (8bits, 16bits …) which depends on the number of tasks to be provided by the scheduler module, n should be choosen at desing stage by the scheduler designer.
The mask is used to mark a task for execution, when the binary counter and the mask: (mask & counter) == mask From the previous definition, the task is assigned to a range of time-slices.
Given the mask and the OS tick period we can obtain the task rate. Therefore the task rate is: task rate = OS tick * (mask + 1).

Offset Concept

A collision may occur between the tasks when the tasks share the same time-slice. If a collision is present some tasks will start being executed in a not desirable time. An offset is defined to allow the task execution being moved in different time-slices. The offset can only be defined in the range from the count of zero up to the value defined by the mask. When the counter matches the mask, and the matched value is the same as the given offset the task is ready to be executed. (mask & counter) == offset With this approach the task collision is avoided.

Dependencies to other modules

The scheduler module has dependencies on project specific timer module.
The include structure of the Scheduler module shall be as follows:



File Name	Description
Tasks.c	Provides the timed task definitions
Tasks.h	Export the timed task interfaces to the scheduler configuration file
Exceptions.c	Setup of IVPR to point to the EXCEPTION HANDLERS memory area defined in the linker command file.
Exceptions.h	Export the Exception handlers to scheduler module
Timer.c	Configurate the STM timer initialization and configurate the STM flag in 1 ms
Timer.h	Export the STM flag and rise the tickflag each 1 millisecond
Leds.c	Configured the PORT A as outputs and led1 to led4 on board
Leds.h	Export the configuration of porta and led1 to led4 on board
Typedesfs.h	This file defines all of the data types for the Motorola header file
Stdtypedef.h	Public type header file for the coreHAL
INIT.c	Initialise PLL before turning it on
INIT.h	Export the pll configuration
IntcInterrupts.c	Contains an implementations of generic interrupt controller handling routines for the MPC56xx and PX MCU families
IntcInterrupts.h	Export the interface of interrupt controller handing
Kernel.h	Contain all headers implemented in the system

API Specification

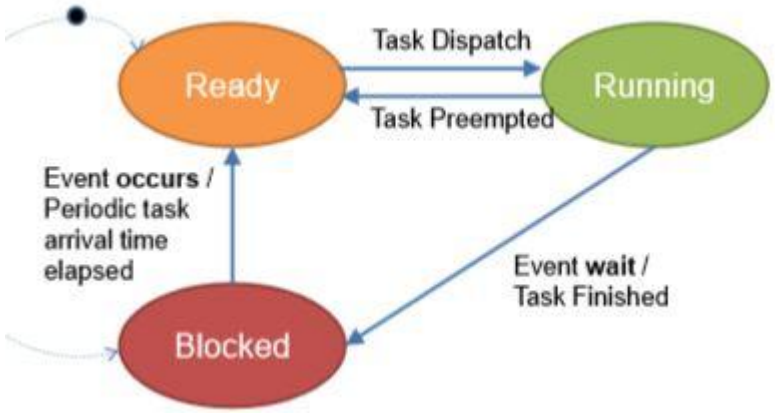
Name:	Taskmasktype			
type	U32			
Range:	tickflag	Mask_1ms	OxFA00	Mask required for 1 ms task
	task1	Mask_1ms		Mask required for 1 ms task
	task2	Mask_3ms		Mask required for 3 ms task
	Task3	Mask_5ms		Mask required for 5 ms task
	task4	Mask_17ms		Mask required for 17 ms task
Description	The mask values to generate the task periods			

STM time formula

$$Time\ Value = \frac{(Time) * Frec}{Prescalator} = \frac{(1x10^{-3}\ seg) * 64MHz}{1} = 64x10^{3}$$

DECIMAL $= 64x10^{3}$ $\longrightarrow$ HEX = 0x0000FA00

Name:	taskStateType	
Type:	U8	
Range:	Task_state_suspended	Task state initial value
	Task_state_ready	Task state indicates the task is ready to be executed
	Task_state_running	Task state indicates the task iscurrectly running
Description:	Task states	



Schmodule_configType

Name:	scheduler	
Type:	u8	
Range:	n.a.	Structure to hold the module 's configuration set. The Contents of this data structure are implementation specific
Description:	Counter tasks trought the tickflag	

Name:	**S_TASKS**	
Type:	structure	
Range:	Implementation specific structure	Structure to hold the module’s configuration set. The contents of this data structure are implementation specific.
Description:	Configuration of members: period, offset and function of the tasks	

Name:	TASK_LIST	
Type:	array	
Range:	Implementation specific array	array to hold the module's configuration set. The contents of this data structure are implementation specific.
Description:	Configuration of members: period, offset and function of the tasks	

Public Function Definitions

Public functions shall be exported in SchModule.h file and defined in SchModule.c file.

Name:	**INTC_InstallINTCInterruptHandler**	
syntax	Void ***INTC_InstallINTCInterruptHandle***r(function, port,1)	
Parameters(in /out)	Function type	handlerFn
Parameter2(in /out)	Port(N)	unsigned short vectorNum Configuration of required port of the board
Parameter3(in /out)		unsigned char psrPriority
Description:	Configuration of interrupts handler	

DESCRIPTION: Contains an implementations of generic interrupt controller handling routines for the MPC56xx and PX MCU families.

Name:	INTC_INTCInterruptHandler	
syntax	Void ***INTC_INTCInterruptHandler*** (function, port,1)	
Parameters(in /out)	n.a.	handlerFn
Description:	e200z0h IVOR branch table interrupts for core 0.	

Private Function Definitions

Private functions shall be defined in SchModule.c file.

Service name:	Tickflag
Syntax:	Void ***tickflag***(Void)
Parameters(in/out)	None
Parameters(in/out)	None
Returns value:	None
Description:	Callback function periodically called from the timer module providing the tick reference

Service name:	STM_config_clock
Syntax:	Void ***STM_config_clock*** (Void)
Parameters(in/out)	None
Parameters(in/out)	None
Returns value:	None
Description:	Configure the STM flag at 1ms and initialize the counter

Service name:	function_time
Syntax:	Void ***function_time*** (Void)
Parameters(in/out)	None
Parameters(in/out)	None
Returns value:	None
Description:	resets the flag on channel 0 and rise the flag each 1 millisecond

Name:	Tareas	
Type:	enum	
Range:	Implementation specific enum	enum to hold the module’s configuration set. The contents of this data structure are implementation specific.
Location:	Tasks.h	
Description:	add the number of tasks	

Adding more functions to the scheduler

In order to add another task in the scheduler system, please follow the next steps:

1) In the file tasks.h, in the ***enum*** structure, write with the same format, the name of the next function. Is important not to modify ***E_ISK_TASK_NUM***. Please refer to the next image:

```
typedef enum {

    E_ISK_TASK1,
    E_ISK_TASK2,
    E_ISK_TASK3,
    E_ISK_TASK4,


    //E_ISK_TASK...., <------ Add the next task with the following number


/*DONT MODIFY THIS LINE*/
    E_ISK_TASK_NUM
} tareas;
```

Example of a new task added:

```
typedef enum {

    E_ISK_TASK1,
    E_ISK_TASK2,
    E_ISK_TASK3,
    E_ISK_TASK4,
    E_ISK_TASK5, //<------ Example of a new task added

    //E_ISK_TASK...., <------ Add the next task with the following number


/*DONT MODIFY THIS LINE*/
    E_ISK_TASK_NUM
} tareas;
```

2) In the file main.c, in the Inline functions section, in the ***S_TASK*** type structure ***TASK_LIST***, write with the same format, the name of the next task, its period and its offset. Please refer to the next image:

```
const S_TASKS TASK_LIST[E_ISK_TASK_NUM]={
        {task1, 200, 1},
        {task2, 400, 3},
        {task3, 800, 5},
        {task4, 1600, 17}
    // {task.., period, offset}...., <------ Add the next task with the following number, its period and its offset
};
```

Example of a new task added:

```
const S_TASKS TASK_LIST[E_ISK_TASK_NUM]={
        {task1, 200, 1},
        {task2, 400, 3},
        {task3, 800, 5},
        {task4, 1600, 17},
        {task5, 1200, 13} //<------ Example of a new task added

    // {task.., period, offset} <------ Add the next task with the following number, its period and its offset
};
```

3) In the task.c file, add the next function for the task with its following number and content.

```
void task3(void)
{
    SIU.GPDO[70].B.PDO ^= 1;//  toggles led3
    SIU.GPDO[2].B.PDO ^= 1;//  toggles external led3
}

void task4(void)
{
    SIU.GPDO[71].B.PDO ^= 1;// toggles led4
    SIU.GPDO[3].B.PDO ^= 1; // toggles external led4
}


/* void task...(void) <------ Add the next task with its following number and content
  {
    FUNCTION CONTENT
  } */
```

Example of a new task added:

```
void task3(void)
{
    SIU.GPDO[70].B.PDO ^= 1;        // toggles led3
    SIU.GPDO[2].B.PDO ^= 1;        //  toggles extern led3 in PORTA
}

void task4(void)
{
    SIU.GPDO[71].B.PDO ^= 1;       // toggles led4
SIU.GPDO[3].B.PDO ^= 1;           // toggles extern led4 in PORTA
}


void task5(void)
{
    SIU.GPDO[71].B.PDO ^= 1;   //<------Example of a new task added
SIU.GPDO[4].B.PDO ^= 1;           // toggles extern led5 in PORTA
}
```